Aufgabe 1 (3 Punkte):
Aufgabe 2 (3 Punkte):
Aufgabe 3 (4 Punkte): Aufgabe 4 (1 Punkt): Aufgabe 5 (3 Punkte):

Familienname:

Aufgabe 6 (1 Punkt):
Aufgabe 7 (2 Punkte):
Aufgabe 8 (3 Punkte): Aufgabe 9 (1 Punkt):

Vorname:

Aufgabe 10 (3 Punkte):
Aufgabe 11 (4 Punkte):
Aufgabe 12 (3 Punkte):

Matrikelnummer:

Aufgabe 13 (3 Punkte): Aufgabe 14 (2 Punkte):
Aufgabe 15 (4 Punkte):

Gesamtpunkte (40 Punkte):

Schriftlicher Test (120 Minuten)
VU Einfuhrung ins Programmieren f¨ ur TM¨

28. Januar 2019

Aufgabe 1 (3 Punkte). Was versteht man unter *Call-by-Value*? Was versteht man unter *Call-byReference*? Was ist der Unterschied in der Praxis? Gibt es beides in C++?

Losung zu Aufgabe 1.¨

Die Parameterübergabe zu Routinen / Methoden / Funktionen erfolgt über 2 Arten:

Call-by-Value (Wertparameter): Eine Kopie wird übergeben. Änderungen haben keinen Einfluss auf das Originale.

Call-by-Referenz (Referenzparameter): Änderungen an den Parameter wirken sich auf das Originale aus.

Im Normallfall wird alles per Call-By-Value übergeben.
Mit Pointers bzw. dem "&" operator wird die Referenzübergabe in der Signatur bestimmt.

In der Praxis werden Objekte als Referenz übergeben. Dies verhindert den Kopiervorgang, das sich auf auf die Performance auswirkt.

Aufgabe 2 (3 Punkte). Schreiben Sie eine rekursive C++ Funktion binomial, die mit Hilfe des Additionstheorems

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1} \qquad \text{fur¨ } k,n \in \mathrm{N} \text{ mit } 2 \le k < n$$

den Binomialkoeffizienten berechnet. Beachten Sie dabei

$$\binom{n}{1} = n \quad \text{und} \quad \binom{n}{n} = 1 \quad \text{fur alle¨ } n \ge 1.$$

Stellen Sie mittels assert sicher, dass fur die Input-Parameter 1¨ ≤ k ≤ n gilt.

Losung zu Aufgabe 2.¨

```
int binomial(int n, int k) {
    assert(1 <= k && k <= n);
    if (k == 1) return n;
    if (n == k) return 1;
    return binomial(n - 1, k) + binomial(n - 1, k - 1);
}
```

Aufgabe 3 (4 Punkte). Was ist der Shell-Output des folgenden Programms?

```
#include <iostream>
using std : : cout ; using
std : : endl ;

class foo {
private : int x ;
int y ;

public :
  foo ( int x , int y) {
    this−>x = x ; this−>y = y ;
    cout << ”create : x=” << x << ” , y=” << y << endl ;
  }

  ˜foo () { cout << ” delete : x=” << x << ” , y=” << y << endl ;
  }

  foo (const foo& input ) {
    int a = input . x ; int b =
    input . y ; int c = 0; while
    ( a != b ) {
      if ( a < b ) { c = a ; a
        = b; b = c ;
      }
      a = a − b;
                    cout << ”a=” << a << ” , b=” << b << ” , c=” << c << endl ;
    }
    x = input . x/a ; y =
    input . y/b;
  }
};


int main() { foo
  q(9 ,33); foo r =
  q ; return 0;
}
```

Losung zu Aufgabe 3.¨

```
create: x=9, y=33
a=24, b=9, c=9
a=15, b=9, c=9
a=6, b=9, c=9
a=3, b=6, c=6
a=3, b=3, c=3
delete: x=9, y=33
delete: x=3, y=11
```

Hinweis. In den folgenden Aufgaben seien die Polynome $p(x) = \sum_{j=0}^{n} a_j x^j$ als Objekte der C++ Klasse Polynomial gespeichert, die unten definiert ist. Neben Konstruktor, Kopierkonstruktor, Destruktor und Zuweisungsoperator, gibt es eine Methode, um den Grad n auszulesen (degree), und eine, um die erste Ableitung von p zu berechnen (diff). Den j-ten Koeffizienten a_j von p erhalt man¨ mittels p[j], den Funktionswert $p(x)$ an einer Stelle $x \in$ R durch p(x).

```
1  class Polynomial {
2  private :
3      int n;
4      double* a ;
5      public :
6      Polynomial ( int n=0);
7      Polynomial (const Polynomial&);
8      ~Polynomial ();
9      Polynomial& operator=(const Polynomial&);
10     int degree () const ;
11     Polynomial        diff () const ;
12     double operator ()( double x) const ;
13     const double& operator [ ] ( int j ) const ;
14     double& operator [ ] ( int j );
15     };
```

Aufgabe 4 (1 Punkt). Erlautern Sie die Bedeutung der beiden¨ const in Zeile 13 der Klassendefinition.

Losung zu Aufgabe 4.¨

const bei Returnwert (bsp.: const double&):

Liefert einen konstanten wert zurück.

Wird beispielsweise verwendet, um temporäre Werte NICHT zu überschreiben.

e.g.: (a*b) = c; // * ist überladener operator

const bei Methode (e.g.: ...methode() const {})

Die Methode hat auf Member der Klasse nur Lesezugriff.

Aufgabe 5 (3 Punkte). Schreiben Sie den Konstruktor der Klasse Polynomial, der den Koeffizientenvektor mit Null initialisiert. Stellen Sie mittels assert sicher, dass $n \geq 0$ ist.

Hinweis. Beachten Sie, dass der Koeffizientenvektor ein Vektor der Lange¨ $n + 1$ ist.

Losung zu Aufgabe 5.¨

```
Polynomial::Polynomial(int n)
{
    assert(n >= 0);
    this->n = n;
    if (n == 0) {
        this->a = new double[1];
        this->a[0] = 0;
    }
    else {
        this->a = new double[n + 1];
        for (int i = 0; i <= n; ++i) {
            this->a[i] = 0;
        }
    }
}
```

Aufgabe 6 (1 Punkt). Schreiben Sie den Destruktor der Klasse Polynomial.

```
Polynomial::~Polynomial()
{
    if (this->a) {
        delete[] this->a;
    }
}
```

Losung zu Aufgabe 6.¨

Aufgabe 7 (2 Punkte). Schreiben Sie den Koeffizientenzugriff der Klasse Polynomial fur¨ constObjekte. Stellen Sie mittels assert sicher, dass der Index $0 \le j \le n$ erfullt, wenn¨ $p(x) = \sum_{j=0}^{n} a_j x^j$ ein Polynom vom Grad n ist. **Losung zu Aufgabe 7.¨**

```
const double& Polynomial::operator[](int j) const
{
    assert(0 <= j && j <= this->n);
    return this->a[j];
}
```

Aufgabe 8 (3 Punkte). Schreiben Sie den Zuweisungsoperator der Klasse Polynomial.
Losung zu Aufgabe 8.¨

```
Polynomial& Polynomial::operator=(const Polynomial& p)
{
    if (this != &p) {
        this->n = p.n;
        if (this->a) delete[] this->a;
        this->a = new double[p.n + 1];
        for (int i = 0; i <= p.n; ++i) {
            this->a[i] = p.a[i];
        }
    }
    return *this;
}
```

Aufgabe 9 (1 Punkt). Es sei $p(x) = \sum_{j=0}^{n} a_j x^j$ ein Polynom vom Grad n. Geben Sie eine explizite Formel fur die erste Ableitung¨ $p^0(x)$ an! Was passiert im Fall $n = 0$?

Losung zu Aufgabe 9.¨

$$p'(x) = \sum_{j=1}^{n} j a_j x^{j-1}$$

Bei n = 0 handelt es sich um eine Konstante. Die Ableitung einer Konstante ist 0.

Aufgabe 10 (3 Punkte). Schreiben Sie die Methode diff der Klasse Polynomial, die die erste Ableitung p^0 eines Polynoms p zuruckgibt. Beachten Sie den Sonderfall, dass¨ p ein Polynom vom Grad $n = 0$ ist.

Hinweis. Das Polynom p soll nicht uberschrieben, sondern ein neues Polynom¨ p erstellt werden.

Losung zu Aufgabe 10.¨

```
Polynomial Polynomial::diff() const
{
    if (this->n == 0) {
        return Polynomial();
    }

    Polynomial p(this->n - 1);
    for (int i = 0; i <= p.n; ++i) {
        p[i] = this->a[i + 1] * (i + 1);
    }
    return p;
}
```

Aufgabe 11 (4 Punkte). Überladen Sie den + Operator so, dass er die Summe $r = p + q$ zweier Polynome $p(x) = \sum_{j=0}^{m} a_j x^j$ und $q(x) = \sum_{k=0}^{n} b_k x^k$ berechnet und zurückgibt. Beachten Sie, dass die Polynome p und q unterschiedlichen Grad $m \neq n$ haben können.

Lösung zu Aufgabe 11.

```
const Polynomial operator+(const Polynomial& p, const Polynomial& q)
{
    int max_n = p.degree() < q.degree() ? q.degree() : p.degree();

    Polynomial r(max_n);
    for (int i = 0; i <= r.degree(); ++i) {
        if (i <= p.degree()) {
            r[i] += p[i];
        }
        if (i <= q.degree()) {
            r[i] += q[i];
        }
    }

    return r;
}
```

Aufgabe 12 (3 Punkte). Uberladen Sie den¨ * Operator so, dass er die Skalarmultiplikationen $r = \lambda p$ bzw. $r = p\lambda$ berechnet und zuruckgibt, wobei¨ $p(x) = \sum_{j=0}^{n} a_j x^j$ ein Polynom ist und $\lambda \in \mathbb{R}$ ein Skalar, d.h. $r(x) = \sum_{j=0}^{n} b_j x^j$ mit $b_j = \lambda a_j$.

Losung zu Aufgabe 12.¨

```
const Polynomial operator*(double c, const Polynomial& p)
{
    Polynomial b(p.degree());
    for (int i = 0; i <= b.degree(); ++i) {
        b[i] = p[i] * c;
    }
    return b;
}

const Polynomial operator*(const Polynomial& p, double c)
{
    return c * p;
}
```

Aufgabe 13 (3 Punkte). Überladen Sie den * Operator so, dass er das Produkt $r = pq$ zweier Polynome $p(x) = \sum_{j=0}^{m} a_j x^j$ und $q(x) = \sum_{k=0}^{n} b_k x^k$ berechnet und zurückgibt.

Hinweis. Beachten Sie, dass r ein Polynom vom Grad $m + n$ ist. Die Koeffizienten von $r(x) = \sum_{\ell=0}^{m+n} c_\ell x^\ell$ sind gerade gegeben durch

$$c_\ell = \sum_{\substack{j+k=\ell \\ j\in\{0,\dots,m\} \\ k\in\{0,\dots,n\}}} a_j b_k.$$

Lösung zu Aufgabe 13.

```cpp
const Polynomial operator*(const Polynomial& p, const Polynomial& q)
{
    Polynomial r(p.degree() + q.degree());
    for (int j = 0; j <= p.degree(); ++j) {
        for (int k = 0; k <= q.degree(); ++k) {
            r[j + k] += p[j] * q[k];
        }
    }
    return r;
}
```

Aufgabe 14 (2 Punkte). Bestimmen Sie den Aufwand Ihrer Funktion aus Aufgabe 13 fur zwei¨ Polynome p und q vom selben Grad n. Falls die Funktion fur¨ $n = 10^2$ eine Laufzeit von 0,5 Sekunden hat, welche Laufzeit erwarten Sie aufgrund des Aufwands fur¨ $n = 10^3$? Begrunden Sie Ihre Antwort!¨

Losung zu Aufgabe 14.¨

Aufwand von 13)

Die Funktion hat 2 for-Schleifen, die jeweils n-mal iteriert werden -> O(n^2).

Beispiel:
$n_1 = 10^4, t_1 = 0{,}5$
$n_2 = 10^6, t_2 = ?$

$$\frac{(n_2)^2}{(n_1)^2} = \frac{(10^6)^2}{(10^4)^2} = 10^2 = 100\ fache\ Aufwand$$

$$t_2 = 100 * 0.5 = 50\ Sekunden$$

Der Aufwand von $n = 10^4$ auf $n = 10^6$ verhundertfacht sich, d.h. der Auwand steigt von 0,5 Sekunden auf 50 Sekunden.

Aufgabe 15 (4 Punkte). Eine Moglichkeit, eine Nullstelle eines Polynoms¨ $p(x)$ zu berechnen, ist das Newton-Verfahren. Ausgehend von einem Startwert x_0 definiert man induktiv eine Folge (x_n) durch $x_{k+1} = x_k - p(x_k)/p^0(x_k)$.

Schreiben Sie eine Funktion root, die zu gegebenem Polynom $p(x)$, Startwert x_0 und Toleranz $\tau > 0$ das Newton-Verfahren durchfuhrt, wobei die Iteration endet, falls¨

$$|p(x_n)| \leq \tau \quad \text{und} \quad |x_n - x_{n-1}| \leq \tau$$

gelten. In diesem Fall werde der letzte Wert x_n als Approximation der Nullstelle zuruckgegeben.¨ **Hinweis.** Verwenden Sie die Klasse Polynomial. Stellen Sie mittels assert sicher, dass $\tau > 0$ gilt. Vermeiden Sie es, alle Folgenglieder zu speichern, weil der Algorithmus ja in jedem Schritt nur die Folgenglieder x_{n-1} und x_n benotigt.¨

```
double root(Polynomial& p, double x, double t)
{
    assert(t > 0);
    Polynomial pd = p.diff();
    cout << p << endl;
    cout << pd << endl;
    double xp = x;                    // x_previous
    double xn = xp - p(xp) / pd(xp);// x_next

    while (fabs(p(xn)) > t || fabs(xn - xp) > t) {
        xp = xn;
        xn = xn - p(xn) / pd(xn);
    }

    return xn;
}
```